你必须知道的 React 生命周期

首先，当面对一些问题:

1 React 用了这么久，经常遇到的问题是 setState 在这里写合适吗?

2 为什么 setState 写在这里造成了重复渲染多次?

3 为什么你的 setState 用的这么乱?

4 组件传入 props 是更新呢？重新挂载呢？还是怎样?

5 ...

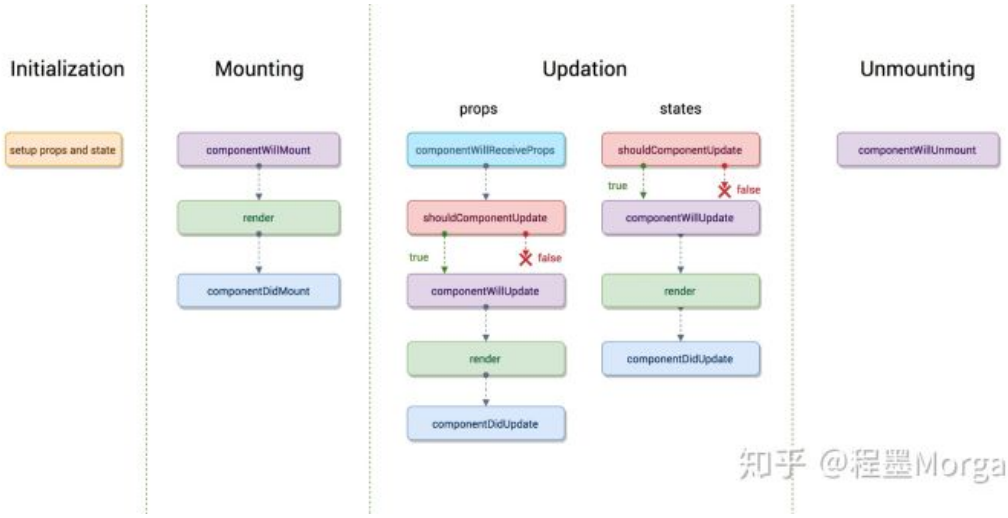
其次，生命周期可以看到整个 Component 的运行过程，在 coding 的时候很自然的找好他们的位置，合作起来就会方便多了，这里极力推荐 airbnb 的 react coding 规范.

React v16.0 刚推出的时候，是增加了一个 componentDidCatch 生命周期函数，这只是一个增量式修改，完全不影响原有生命周期函数；但是，到了 React v16.3，大改动来了，引入了两个新的生命周期函数:

- getDerivedStateFromProps
- getSnapshotBeforeUpdate

首先当然要吐槽的是 React 组件生命周期函数名都好长，完全程序员手打，真的很容易犯错，不过，为了语义清晰嘛，也可以理解，而且我们应该都有各种方法在代码编辑器中自动补齐，避免打字打错的情况。而且，这次新的 API getDerivedStateFromProps 实际上就是用来取代以前的函数 componentWillReceiveProps，getDerivedStateFromProps 要比 componentWillReceiveProps 少打两个字符，不亏:-)

先来看 React v16.3 之前的生命周期函数（图中实际上少了 componentDidCatch)，如下图。



这个生命周期函数非常的对称，有 componentWilUpdate 对应 componentDidUpdate，有 componentWillMount 对应 componentDidMount；也考虑到了因为父组件引发渲染可能要根据 props 更新 state 的需要，所以有 componentWillReceiveProps。

但是，这个生命周期函数的组合在 Fiber 之后就显得不合适了，因为，**如果要开启 async rendering，在 render 函数之前的所有函数，都有可能被执行多次。**长期以来，原有的生命周期函数总是会诱惑开发者在 render 之前的生命周期函数做一些动作，现在这些动作还放在这些函数中的话，有可能会被调用多次，这肯定不是你想要的结果。

总有开发者问我，为什么不在 componentWillMount 里写 AJAX 获取数据的功能，他们的观点是，componentWillMount 在 render 之前执行，早一点执行早得到结果。要知道，在 componentWillMount 里发起 AJAX，不管多快得到结果也赶不上首次 render，而且 componentWillMount 在服务器端渲染也会被调用到（当然，也许这是预期的结果），这样的 IO 操作放在 componentDidMount 里更合适。在 Fiber 启用 async render 之后，更没有理由在 componentWillMount 里做 AJAX，因为 componentWillMount 可能会被调用多次，谁也不会希望无谓地多次调用 AJAX 吧。

道理说了都明白，但是历史经验告诉我们，**不管多么地苦口婆心教导开发者不要做什么不要做什么，都不如直接让他们干脆没办法做。**

随着 getDerivedStateFromProps 的推出，同时 deprecate 了一组生命周期 API，包括：

- componentWillReceiveProps
- componentWillMount
- componentWillUpdate

可以看到，除了 shouldComponentUpdate 之外，render 之前的所有生命周期函数全灭，就因为太多错用滥用这些生命周期函数的做法，预期追求对称的美学，不如来点实际的，让程序员断了在这些生命周期函数里做些不该做事情的念想。

至于 shouldComponentUpdate，如果谁还想着在里面做 AJAX 操作，那真的是没救了。

按照官方说法，以前需要利用被 deprecate 的所有生命周期函数才能实现的功能，都可以通过 getDerivedStateFromProps 的帮助来实现。

这个 getDerivedStateFromProps 是一个静态函数，所以函数体内不能访问 this，简单说，就是应该一个纯函数，纯函数是一个好东西啊，输出完全由输入决定。

```
static getDerivedStateFromProps(nextProps, prevState) {
  //根据 nextProps 和 prevState 计算出预期的状态改变，返回结果会被送给 setState
}
```

看到这样的函数声明，应该感受到 React 的潜台词：**老实做一个运算就行，别在这里搞什么别的动作。**

每当父组件引发当前组件的渲染过程时，getDerivedStateFromProps 会被调用，这样我们有一个机会可以根据新的 props 和之前的 state 来调整新的 state，如果放在三个被 deprecate 生命周期函数中实现比较纯，没有副作用的话，基本上搬到 getDerivedStateFromProps 里就行了；如果不幸做了类似 AJAX 之类的操作，首先要反省为什么自己当初这么做，然后搬到 componentDidMount 或者 componentDidUpdate 里面去。

所有被 deprecate 的生命周期函数，目前还凑合着用，但是只要用了，开发模式下会有红色警告，在下一个大版本（也就是 React v17）更新时会彻底废弃。

React v16.3 还引入了一个新的声明周期函数 getSnapshotBeforeUpdate，这函数会在 render 之后执行，而执行之时 DOM 元素还没有被更新，给了一个机会去获取 DOM 信息，计算得到一个 snapshot，这个 snapshot 会作为 componentDidUpdate 的第三个参数传入。

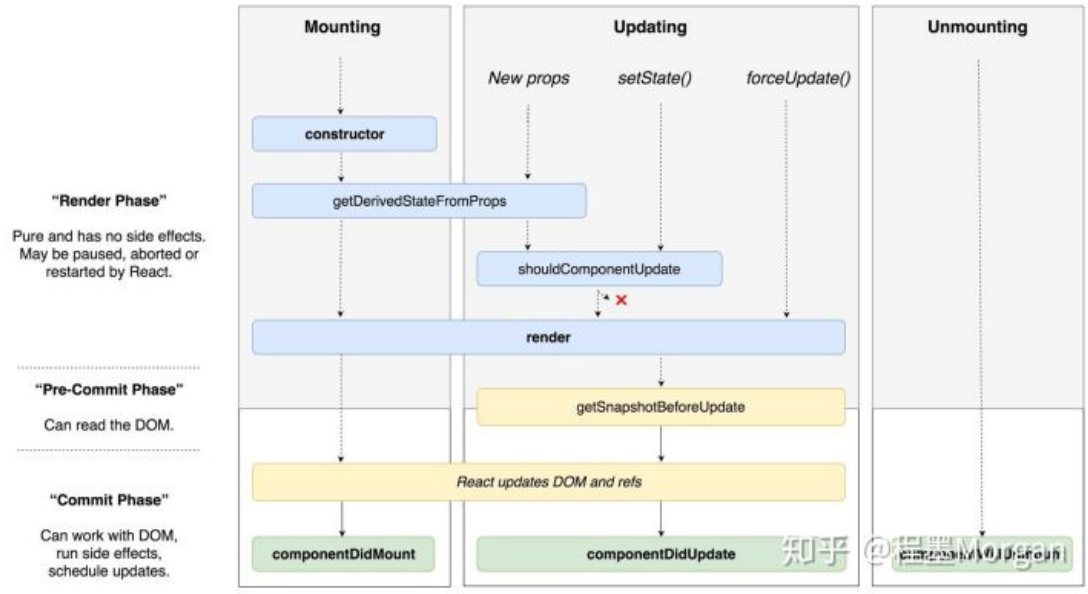
```
  getSnapshotBeforeUpdate(prevProps, prevState) {
    console.log('#enter getSnapshotBeforeUpdate');
    return 'foo';
  }

  componentDidUpdate(prevProps, prevState, snapshot) {
    console.log('#enter componentDidUpdate snapshot = ',
snapshot);
  }
```

上面这段代码可以看出来这个 snapshot 怎么个用法，snapshot 咋看还以为是组件级别的某个“快照”，其实可以是任何值，到底怎么用完全看开发者自己，getSnapshotBeforeUpdate 把 snapshot 返回，然后 DOM 改变，然后 snapshot 传递给 componentDidUpdate。

官方给了一个例子，用 getSnapshotBeforeUpdate 来处理 scroll，坦白说，我也想不出其他更常用更好懂的需要 getSnapshotBeforeUpdate 的例子，这个函数应该大部分开发者都用不上（听得懂我的潜台词吗：**不要用**！）

所以，React v16.3 之后的生命周期函数一览图成了这样。



可以注意到，说 getDerivedStateFromProps 取代 componentWillReceiveProps 是不准确的，因为

componentWillReceiveProps 只在 Updating 过程中才被调用，而且只在因为父组件引发的 Updating 过程中才被调用（往上翻看第一个图）；而 getDerivedStateFromProps 在 Updating 和 Mounting 过程中都会被调用。

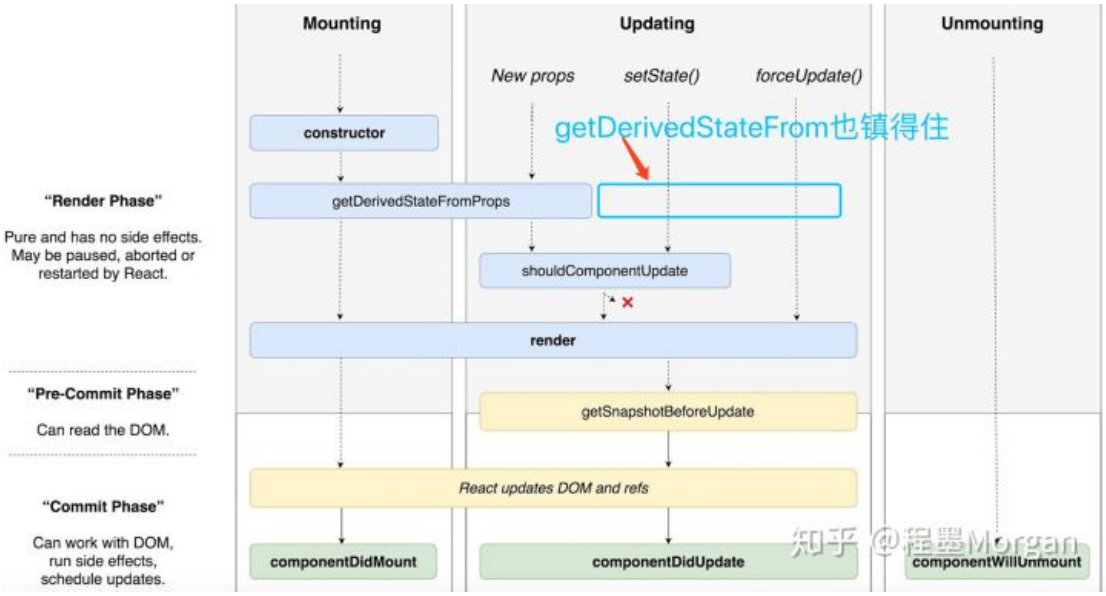
此外，从上面这个也看得出来，同样是 Updating 过程，如果是因为自身 setState 引发或者 forceUpdate 引发，而不是不由父组件引发，那么 getDerivedStateFromProps 也不会被调用。

这其实容易引发一些问题，不用仔细想，光是由此让开发者不得不理解这乱七八糟的差异，就可以知道这是一个大坑！

还好，React 很快意识到这个问题，在 React v16.4 中改正了这一点，改正的结果，就是让 getDerivedStateFromProps 无论是 Mounting 还是 Updating，也无论是因为什么引起的 Updating，全部都会被调用。

这样简单多了！

所以，上面的生命周期函数一览图要改一改。



修正后的生命周期函数图

总结一下：

用一个静态函数 getDerivedStateFromProps 来取代被 deprecate 的几个生命周期函数，就是强制开发者在 render 之前只做无副作用的操作，而且能做的操作局限在根据 props 和 state 决定新的 state，而已。

这是进一步施加约束，防止开发者乱来，我说过，施加约束的哲学指导思想，是我最爱 React 的原因。